DAT328
SQL Server 2005: Index Creation Best Practices

Kimberly L. Tripp
President, SQLskills.com

Microsoft

Speaker – Kimberly L. Tripp

- Independent Consultant/Trainer/Speaker/Writer
- Founder, SYSolutions, Inc. www.SQLskills.com

  email: Kimberly@SQLskills.com

  Become a subscriber on SQLskills.com and learn about new resources which can improve your productivity and server performance!
- Microsoft Regional Director http://www.microsoftregionaldirectors.com/Public/
- SQL Server MVP http://mvp.support.microsoft.com/
- Author for some SQL Server 2005 Whitepapers on MSDN (links from www.SQLskills.com)
- Coauthor MSPress: SQL Server 2000 High Availability, Presenter/Technical Manager for SQL Server 2000 High Availability DVD
- Writer/Editor for SQL Magazine www.sqlmag.com

Microsoft

Overview

- Index Usage and Effective Queries
- Index Concepts
- Table Structures: Internals
- Clustered Tables: The Pros and Cons
- Non-clustered Indexes: The Pros and Cons
- Finding the Right Balance
- Adding Non-Clustered Indexes for Better Performance
- Keeping Performance Optimal

Index Usage
Conceptual

- Allows faster access to data, improving:
  - Lookup/query time
  - Insert/Update time – record location defined
  - Delete time – record location also defined
- Con: overhead for modifications
- Index Strategy Concepts
  - Fewer indexes are better than lots of indexes
  - Wider indexes have more uses
    - Can be used for point queries
    - Can be used by NARROW low selectivity queries
- Write Effective Queries to better take advantage of indexes…

Accessing Data: Limit Data

- Subset of columns = Projection
  - Do not use * (unless against view)
  - Optimizer has more chances for optimizing query when result set is NARROW (only the required columns)
- Subset of rows = Selection
  - Use positive search arguments
  - Isolate the column to one side of the expression
    - USE: MonthlySalary > *value*/12 (constant, seekable)
    - DO NOT USE: MonthlySalary * 12 > *value* (must scan)
  - Be cautious with LEADING wildcards
    - USE: LastName LIKE 'S%'
    - Try not to just append %val% to every application
- Consider using Views, Stored Procedures and Functions to limit the columns/rows

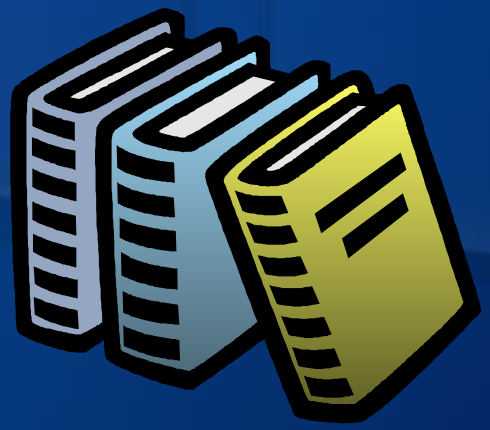
Index Concepts
Book analogy

- Think of a book—with indexes in the back
- The book has one form of logical ordering
- For references, you use the indexes in the back…to find the data in which you are interested you look up the key
- When you find the key, you must lookup the data based on its location… i.e., a “bookmark” lookup
- The bookmark always depends on the (book) content order

Index – Species Common Name

Index – Animal by Type, Name *Bird, Mammal, Reptile, etc…*

Index – Animal by Country, Name



Index – Animals by Habitat, Name *Air, Land, Water*

Index – Species Scientific Name

Index – Animal by Continent, Country, Name

Microsoft

Index Concepts
Tree analogy

- If a tree were data and you were looking for leaves with a certain property, you would have two options to find that data:
- 1) Touch every leaf – interrogating each one to determine if they held that property…SCAN
- 2) If those leaves (which had that property) were grouped such that you could start at the root, move to the branch and then directly to those leaves…SEEK

Index Usage
Physical

- "Data" is stored in Index
  - Leaf level of an index (regardless of index type) stores relevant data for *every* row of the table
    - Clustered Index Leaf Level = Data
    - Non-clustered Index Leaf Level
      - Without additional "include" columns
        - Stores the NC Index KEY + the CL Key (or RID of Heap)
      - With additional "include" columns
        - Stores the NC Index KEY + additionally included columns + the CL Key (or RID of Heap)
  - Non-leaf level(s) are for navigation
- Indexes can be used to answer a query OR to help "point" to full data row to answer query

INCLUDE Non-Key Columns
New in SQL Server 2005

- Leaf level of index can include non-key columns
- Index key limited to 900 bytes/16 columns – this is to keep tree structure and non-leaf levels optimal/small
- Allows more covering indexes
- Allows LOB data types (use sparingly)
- Also, consider Indexed Views for more interesting/wider index options

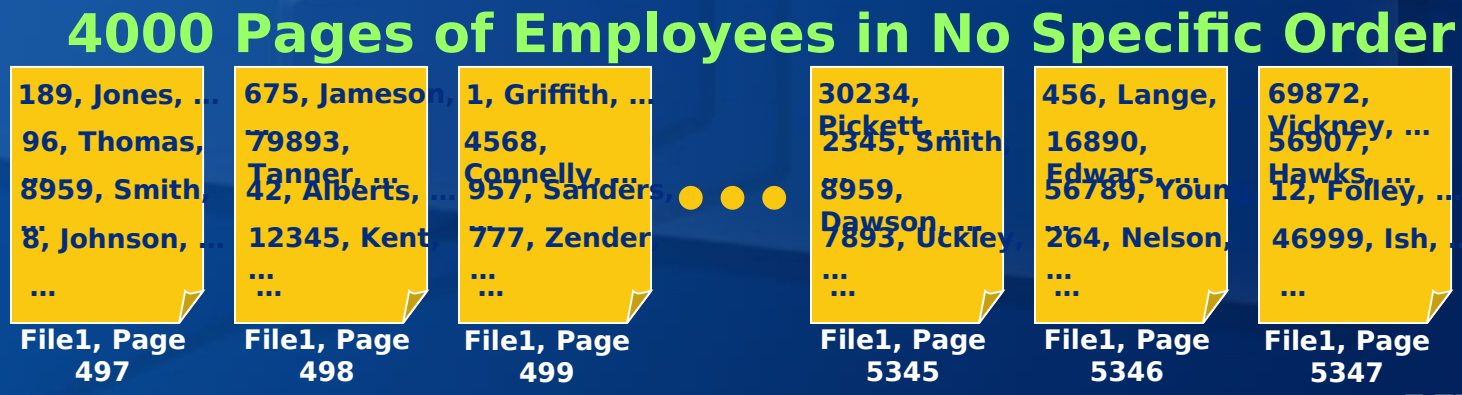
Table Structure

- HEAP: A table without a clustered index
- Clustered Table: A table with a clustered index
- Non-clustered Indexes *do not* affect the base table's structure
- However, Non-clustered Indexes are affected by whether or not the table is Clustered…

Hint: The non-clustered index dependency on the clustered index should impact your choice for the clustering key!

Table Structure: Heap

- Table without a Clustered Index
- Records are *not ordered*
- No Doubly-Linked List
- Access via Index Allocation Map (IAM)
  - IAMs = 8 K Page (Chain) which Tracks Object Usage
    - 1 - IAM per Table/Index (for inrow_data), per File, per 4 GB
- If *no* Indexes exist – a full Table Scan required
  At least 4000 I/Os on the Employee Table Heap



Microsoft

Heap: Pros

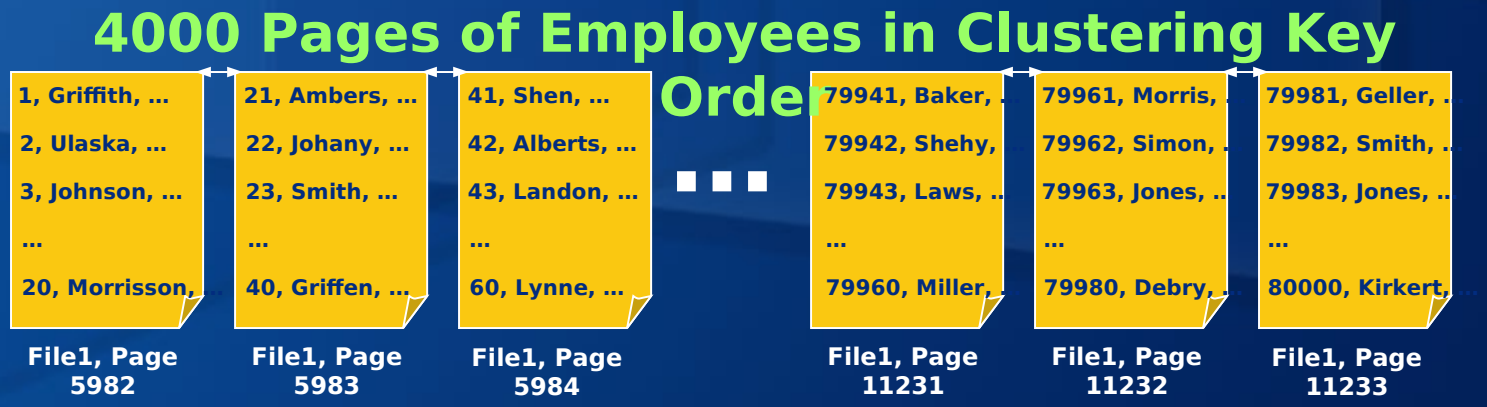
- Effective for “staging” data
  - Truncate table
  - Drop non-clustered Indexes
  - Drop CL Index
  - Load data (consider parallel bulk load)
  - Create CL Index, create non-clustered Indexes
- Excellent for “staging” data into a table that will become a partition to a partitioned table (SQL Server 2005) or to a partitioned view (SQL Server 2000)
- For more on Partitioning, read MSDN whitepaper: SQL Server 2005 Partitioned Tables http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql90/html/sql2k5partition.asp

Heap: Pros

- Indexes can be created after load and index creation can be parallelized
- Efficient for SCANs ONLY – when no UPDATEs (otherwise, forwarding pointers – scans become significantly less efficient)
- Space Efficient – Space from Deletes is reused on subsequent Inserts (at the cost of performance)
- Best used when table is not typical OLTP or combo OLTP/DSS table
- Forward only “logging” table

Table Structure
Clustered table

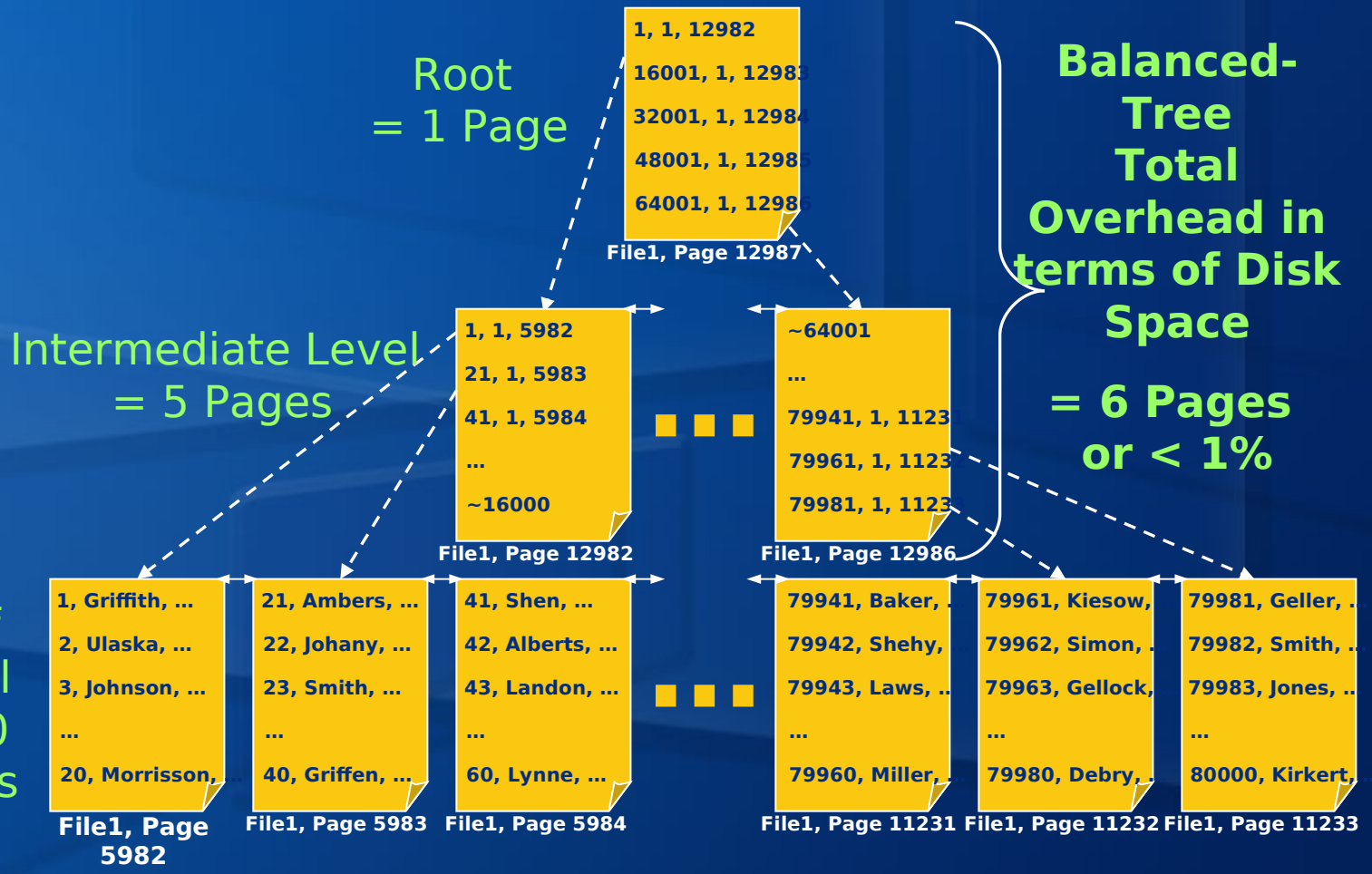
- Clustered Index defines order – applied at CREATION
- Expensive—in time and space—to build
  (an additional 1-2 times the index size for (re)build)
- Table is a doubly-linked list – order maintained LOGICALLY
- *Might* be expensive to maintain



Clustered EmployeeID

Employee Table

Consider a clustered index on an identity column (even better if this is the table's Primary Key)

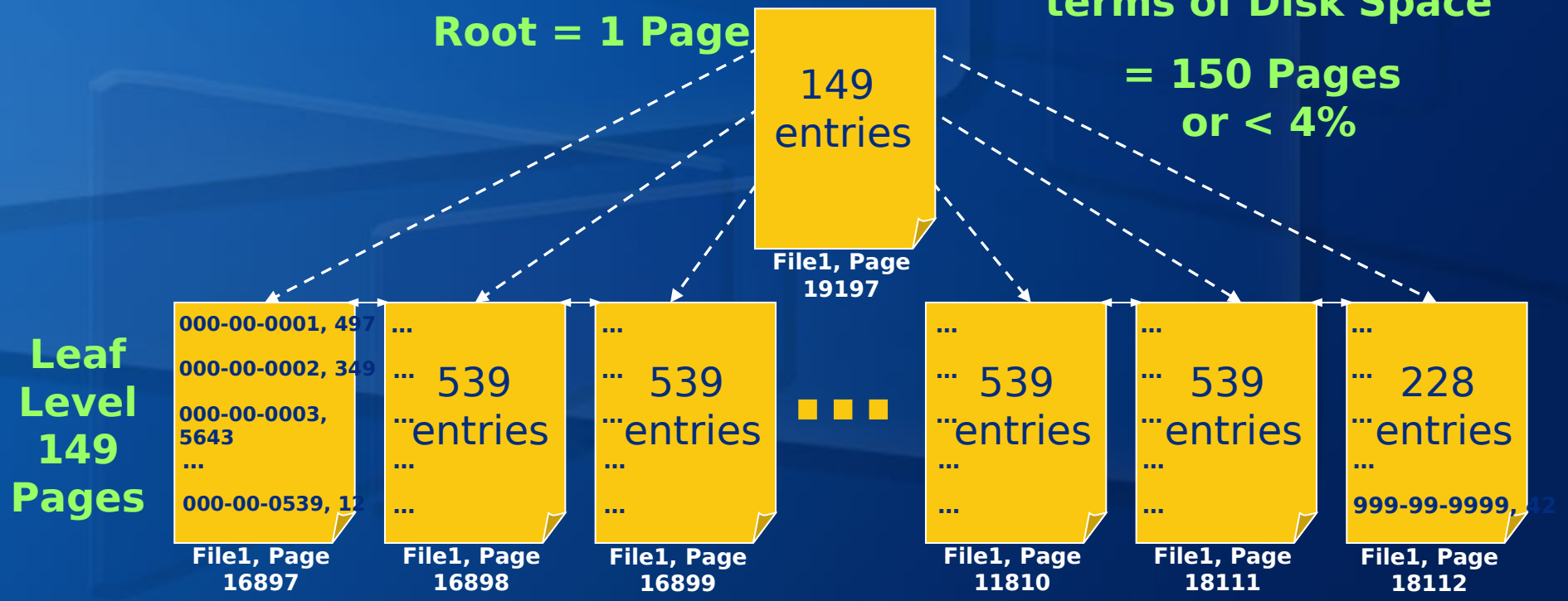


Non-Clustered Indexes
Physical

- Depend on whether the table is a Heap or Clustered
- Clustered Table
  - Rows use the Clustering Key
  - No additional OH to add this column – may use actual data to define the clustering key
  - Minimizes NC Index manipulation if rows move (the NC still points to the CL Key)
  - Clustering key should be static and narrow
- Heap
  - Generally, not recommended

Non-Clustered Index
Unique Key SSN

- Leaf level contains the non-clustered key(s) to define the order
- Also includes either the Heap's Fixed RID or the Table's Clustering Key



Clustered Index Criteria

- Unique
  - Yes – No overhead, data takes care of this criteria
  - NO – SQL Server must “uniquify” the rows on INSERT. This costs time and space. Each duplicate has a 4-byte uniquifier. In SQL Server 2000, all uniquifiers are regenerated when the CL index is rebuild. This is not true in SQL Server 2005.
- Narrow
  - Yes – Keeps the NC indexes narrow
  - NO – Possibly wastes space
- Static
  - Yes – Improves Performance
  - NO – Costly to maintain during updates to the key especially if row movement and/or splits

In fact, an identity column that’s ever-increasing is ideal…

Or date, identity can also be excellent choice – esp. in partitioned scenario

Clustering on an Identity

- Pros
  - Naturally Unique
    (although not guaranteed – should combine with key constraint)
  - Naturally Static
    (although should be enforced through permissions and/or trigger)
  - Naturally Narrow
    (only numeric values possible, whole numbers with scale = 0)
  - Naturally creates a hot spot…
    - Needed pages for INSERT already in cache
    - Minimizes cache requirements
    - Helps reduce fragmentation due to INSERTs
    - Helps improve availability by naturally needing less defrag
- Con
  - Not appropriate for extremely HIGH Insert volume (400+/sec for one table)
  - Consider a couple of hot spots instead of just one

Optimal Table Structures
Poor clustered index choice

- Worst base table structure is a poor choice for Clustered Index, for example: LastName
  - Fragmentation → poor performance
  - Non-unique → poor performance
    - Uniquifier wastes space and time
    - SQL 2000 only, rebuilding a clustered index on a non-unique clustered index forces non-clustered indexes to be rebuilt
  - Volatile → poor performance
    - Most-duplicated value
    - Could cause record relocation and therefore fragmentation
  - Wide → poor performance
    - Wastes space/time – wide keys take longer to maintain/insert.
    - The wider the key the wider the non-clustered indexes – often with little benefit (although debatable as the non-clustered indexes will cover more queries)
- TIP: Don’t do this! ☺

Optimal Table Structures (cont'd)

Heap is better

- Better base table structure is a HEAP
  - No data row Fragmentation → forwarding rows
  - Could have LOB fragmentation (new LOB_Compaction can help this type of fragmentation)
    - Wastes space in non-clustered indexes by using fixed-RID
    - Scans (due to locking/consistency requirements) can be more expensive!
    - Optimized for space over speed
      - Inserts are slower re: IAM and PFS lookup (reduces the "Swiss cheese problem")
      - Updates can be slower with wide modification – same reason that inserts are slower: record relocation
- TIP: Create HEAPs for staging tables as an interim table for high performance data loading!
- See presentation "High Performance Data Loading" on www.SQLDev.Net for more tips!

Microsoft

Optimal Table Structures (cont'd)

The right clustered index

- Best base table structure is the RIGHT Clustered Index
  - Cluster on identity column or possibly add one for clustering
  - Cluster on composite key (date, identity) for tables that also have this pattern
  - Key Criteria: Static, Narrow and Unique
  - Effective Hot Spot(s) using Identity (or date, id) or composite key to create multiple (but not random) hot spots – possibly consider partitioning for large tables/high volume OLTP

TIP: If this doesn't exist consider adding a surrogate column solely to cluster on it!

TIP: Make sure to find the RIGHT Clustered index for *your* environment

TIP: Make sure to AUTOMATE check on fragmentation and defragment if/when necessary

Clustering for Performance

- Cluster for internal benefits which lead to:
  - Better insert/update performance re: fewer splits
  - Better Availability as full table maintenance may not be needed
- Reliance on Nonclustered indexes is greater:
  - Faster access to narrow/low selectivity range queries
  - NC Indexes are used in numerous non-obvious ways (multiple NC Indexes can be joined to cover a query)
  - More flexible in the definition (i.e., Indexed Views can include computations, substrings, etc.)
  - New INCLUDE clause can allow wider covering indexes!
  - Non-clustered indexes are easier/faster to rebuild (re: smaller) when they become fragmented
  - Non-clustered indexes are easier to keep less fragmented – i.e., rebuilds with fillfactor has greater benefit since the index row is narrower

Finding the Right Balance

- Start with a minimal number of indexes
  - Clustered Index
  - Primary Key (nonclustered, if not clustered)
  - Unique Keys
- Manually index foreign keys
  - Non-unique indexes
  - Speed up join performance
- Use Database Tuning Advisor
- Manually index based on either:
  - Specific query tuning where DTA didn’t help
  - Query frequency

Finding the Right Balance (cont'd)

SQL Server 2000 resources with GREAT best practices – that STILL apply!

- Support Webcast: *Indexing for Performance: Finding the Right Balance* (recorded 11 June 2004)
  - http://msevents.microsoft.com/CUI/EventDetail.aspx?EventID=1032254503&Culture=en-US
- Support Webcast: *Indexing for Performance: Proper Index Maintenance* (recorded 19 July 2004)
  - http://msevents.microsoft.com/CUI/EventDetail.aspx?EventID=1032256511&Culture=en-US

Microsoft

Finding the Right Balance (cont'd)

SQL Server 2000 resources with GREAT best practices – that STILL apply!

- Support Webcast: *SQL Server 2000 Profiler: What's New and How to Effectively Use It*

  http://support.microsoft.com/default.aspx?scid=%2Fservicedesks%2Fwebcasts%2Fwc111400%2Fwcblurb111400%2Easp

- Whitepaper: "Index Tuning Wizard for Microsoft SQL Server 2000"
  - http://msdn.microsoft.com/library/en-us/dnsql2k/html/itwforsql.asp?frame=true

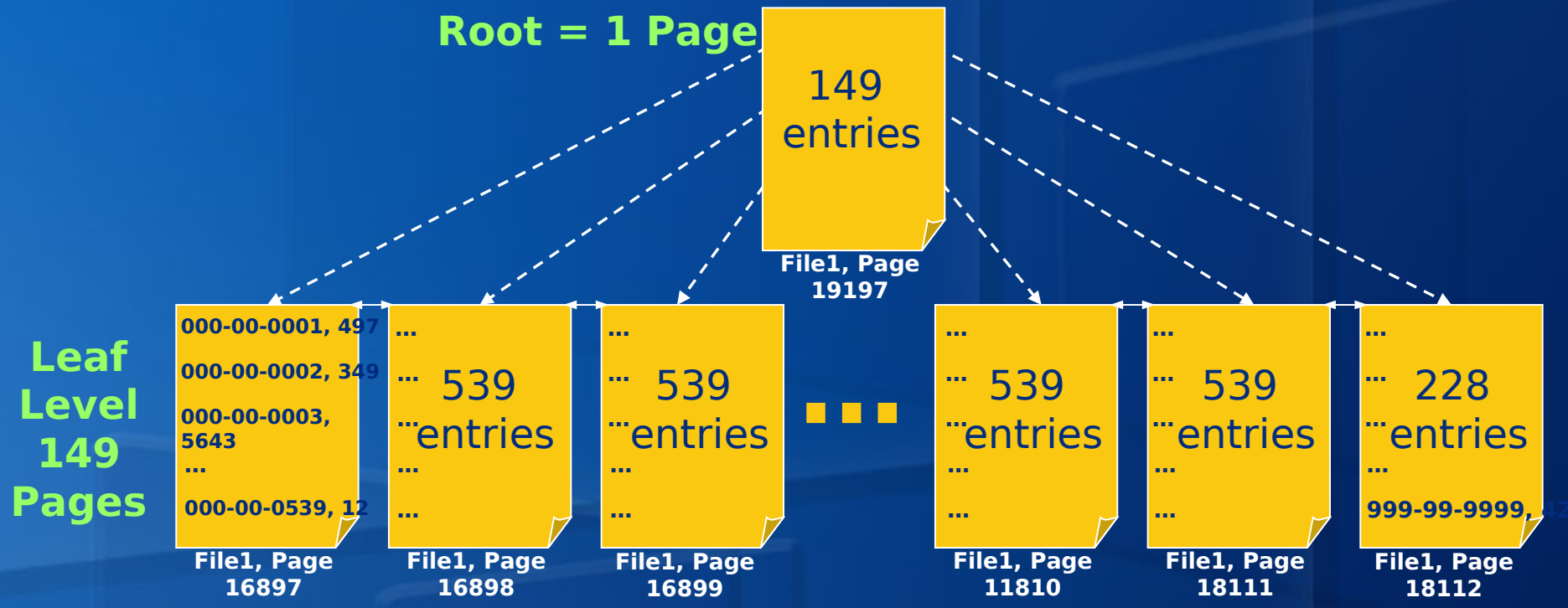
Microsoft

Non-Clustered Indexes

- Internally, non-clustered indexes always store an entry for EVERY row of the table – effectively, they are a “mini” clustered table (of only the index columns)
- If the columns requested in the query are IN the index (regardless of order) then the index COVERs the query
- If queries are narrow this is more likely to happen
- Real-world queries…for example, accessing a large number of tables in a join are typically narrow (the join condition, maybe a search argument and possibly one or two columns in the select list)
- Is it likely that you would ever execute:

SELECT * FROM 8-table-join

What kind of structure is this?



- A non-clustered index on SSN on a table that has a clustered index on EmpID (as we've seen)
- A clustered in on SSN n a table that has ONLY two columns, SSN and EmpID

NOTE: A non-clustered index is EXACTLY the same as a clustered index – but only on the columns that make up the NC index (with the RID

Seems Like…

- Non-clustered indexes are only useful for relatively selective queries…
- What about the following queries?

```
SELECT EmpID, SSN
FROM Employee
WHERE SSN between x and y
```

or

```
SELECT EmpID, SSN
FROM Employee
WHERE EmpID < 10000
```

Think about the access patterns:
- Table Scan (always an option)
- Index Seek w/partial scan
  The nc index on SSN "covers" the query

Think about the access patterns:
- SARG is on clustering key

  The nc index on SSN "covers" the query

THIS SLIDE INTENTIONALLY LEFT BLANK.

Microsoft

Non-Clustered Indexes
For optimal query performance

- Non-Clustered Indexes are BETTER for low selectivity/range queries because:
  - Non-clustered indexes ARE EXACTLY the same as a clustered index—but narrower
  - If your query only needs columns IN an index then the index is said to “cover” your query
  - No – this does *not* mean that you need to cover every query!!!
  - What it does mean: You can make one or two existing indexes slightly wider and you probably will not hurt INSERT, UPDATE, and DELETE performance and may RADICALLY improve SELECT performance
- Non-Clustered Indexes are narrower and easier to maintain (let ITW/DTA help…)
- Use INCLUDE to create narrower trees, more relevant leaf levels for covering!

Covering an Aggregate

- Increasing Gains with different types of indexes can always be achieved
- Each query can be isolated and tuned as if it were in a sandbox – *not* the goal
- Finding the right balance starts with:
  - Base Indexes (CL, PK, UK, FKs)
  - Capturing a workload, tuning with DTA
  - Prioritizing the queries that are still performing poorly and then choose the optimal level of gain by understanding the trade-offs

Indexing for Aggregations

- Two types of Aggregates:
  Stream and Hash
- Try to Achieve Stream to Minimize Overhead in temp table creation
- Computation of the Aggregate Still Required
- Lots of Users, Contention and/or Minimal Cache can Aggravate the problem!

Aggregate Query

- Member has 10,000 Rows
- Charge has 1,600,000 Rows

```
SELECT c.member_no AS MemberNo,
       sum(c.charge_amt) AS TotalSales
FROM dbo.charge AS c
GROUP BY c.member_no
```

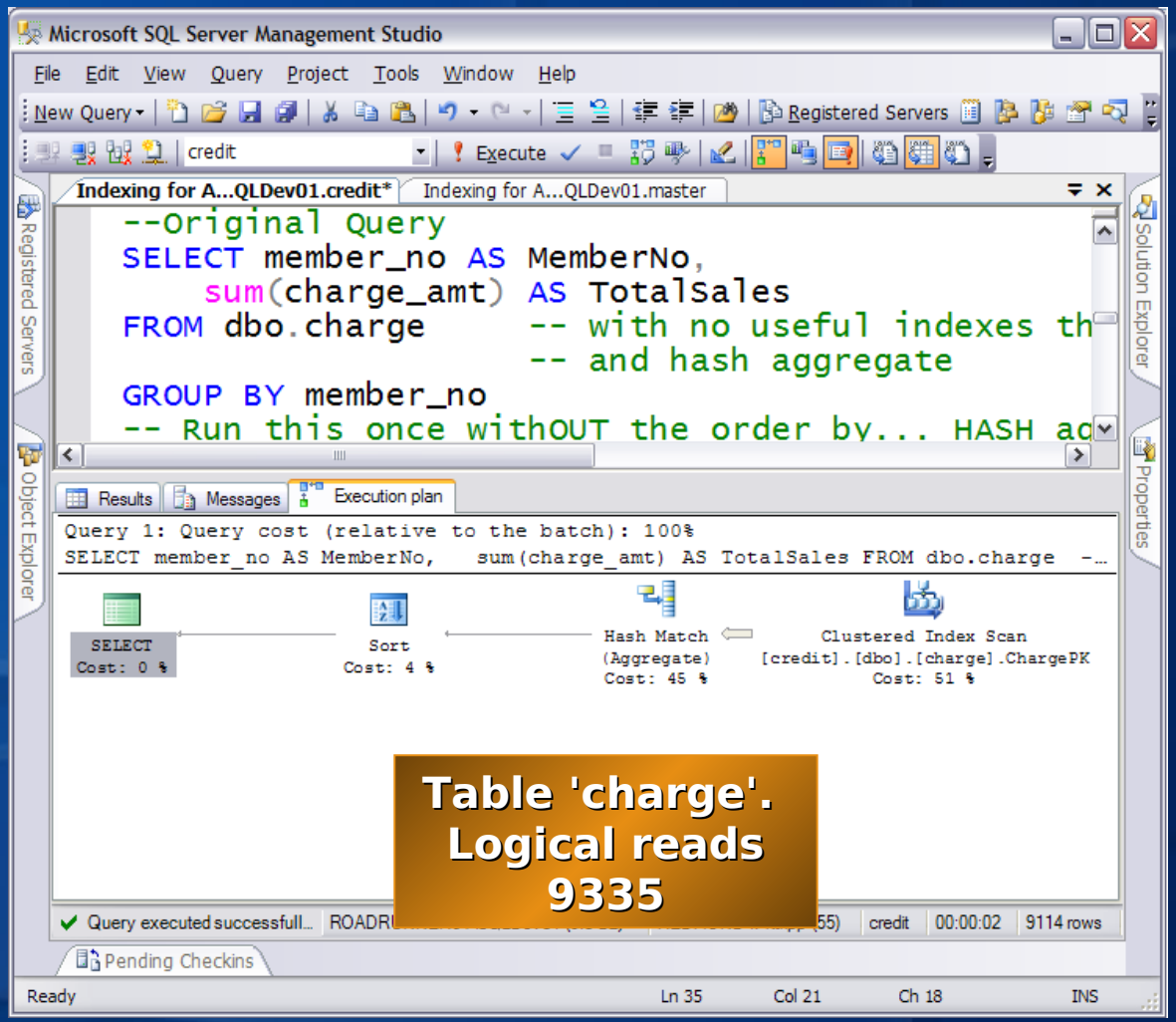
Aggregate Query (cont'd)
Table scan + hash aggregate

```
SELECT c.member_no AS MemberNo,
       sum(c.charge_amt) AS TotalSales
FROM dbo.charge AS c
GROUP BY c.member_no
```

- Table Scan of Charge Table
  - Largest structure to evaluate
  - Worst case scenario
- Worktable created to store intermediate aggregated results – OUT OF ORDER (HASH)
- Data Returned OUT OF ORDER – unless ORDER BY added
- Additional ORDER BY causes another step for SORT – sorting can be expensive!

Worst Case

- Clustered Index Scan
  (table scan)
  1,600,000 rows
- Hash Aggregate
  yields 9,114 rows
  out of order
- Sort
  only has to sort
  9,114 rows instead
  of 1,600,000 rows
- Return Data



Microsoft

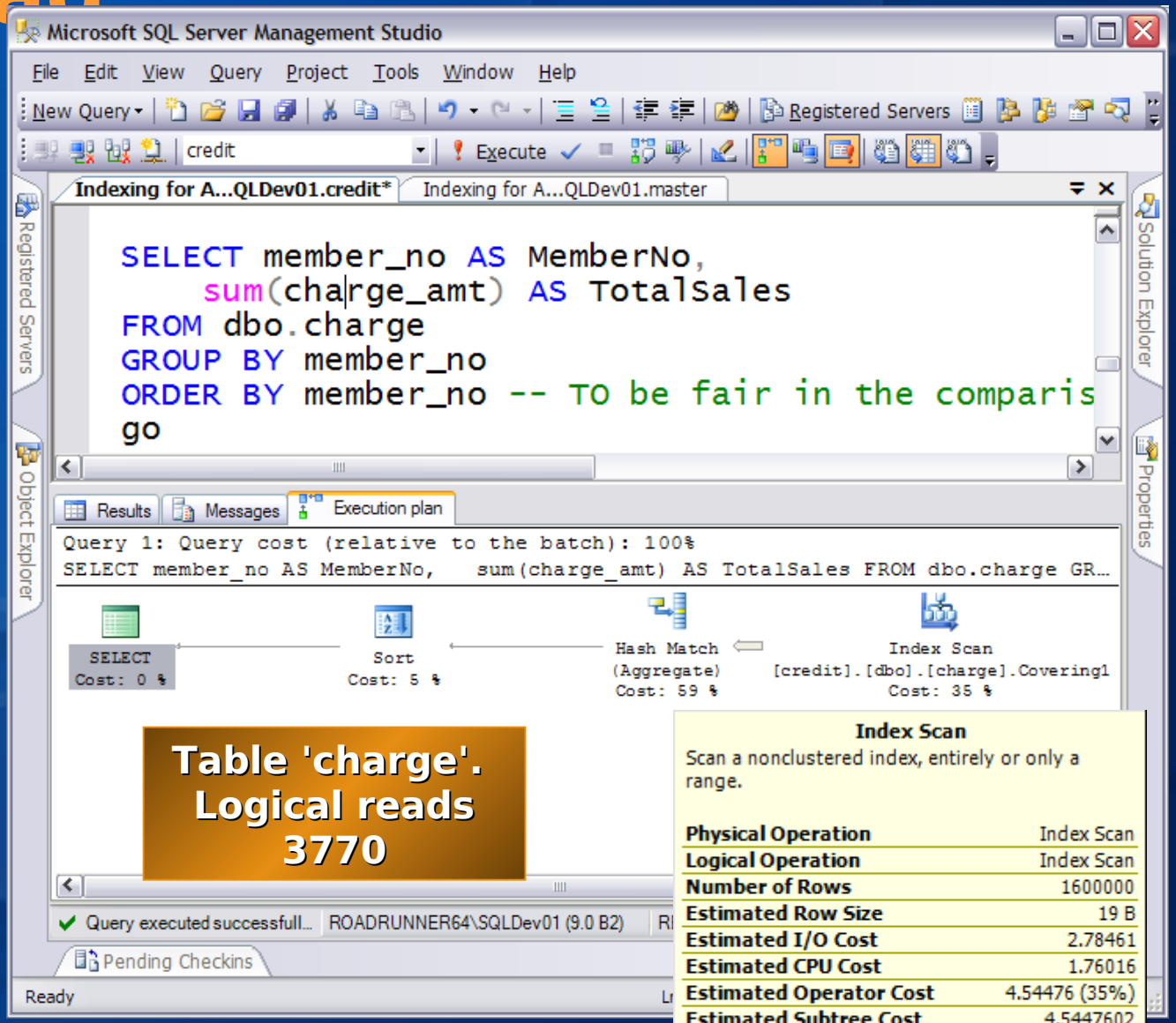
Aggregate Query
Index scan + hash aggregate

```
SELECT c.member_no AS MemberNo,
       sum(c.charge_amt) AS TotalSales
FROM dbo.charge AS c
GROUP BY c.member_no
```

- Out of Order Covering Index on Charge Table
  - Index Exists which is narrower than base table
  - Used instead of table – to cover the query
- Worktable still created to store intermediate aggregated results – OUT OF ORDER (HASH)
- Data Returned OUT OF ORDER – unless ORDER BY added
- Additional ORDER BY causes another step for SORT – sorting can be expensive!

Not as Bad

- COVERING Index Scan
  1,600,000 narrower rows
- Hash Aggregate
  yields 9,114 rows out of order
- Sort
  only has to sort 9,114 rows instead of 1,600,000 rows
- Return Data



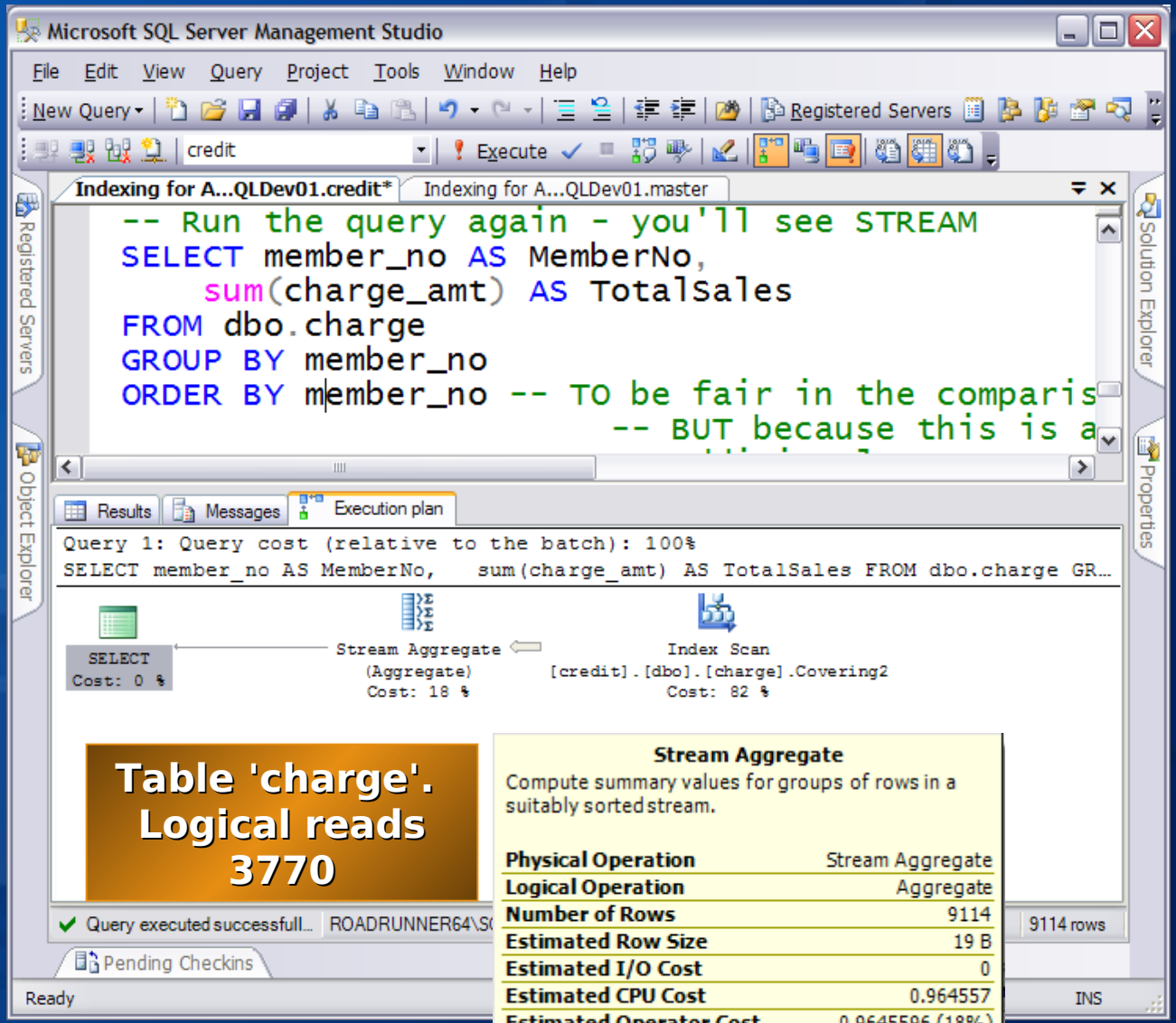
Aggregate Query
Index scan + stream aggregate

```
SELECT c.member_no AS MemberNo,
       sum(c.charge_amt) AS TotalSales
FROM dbo.charge AS c
GROUP BY c.member_no
```

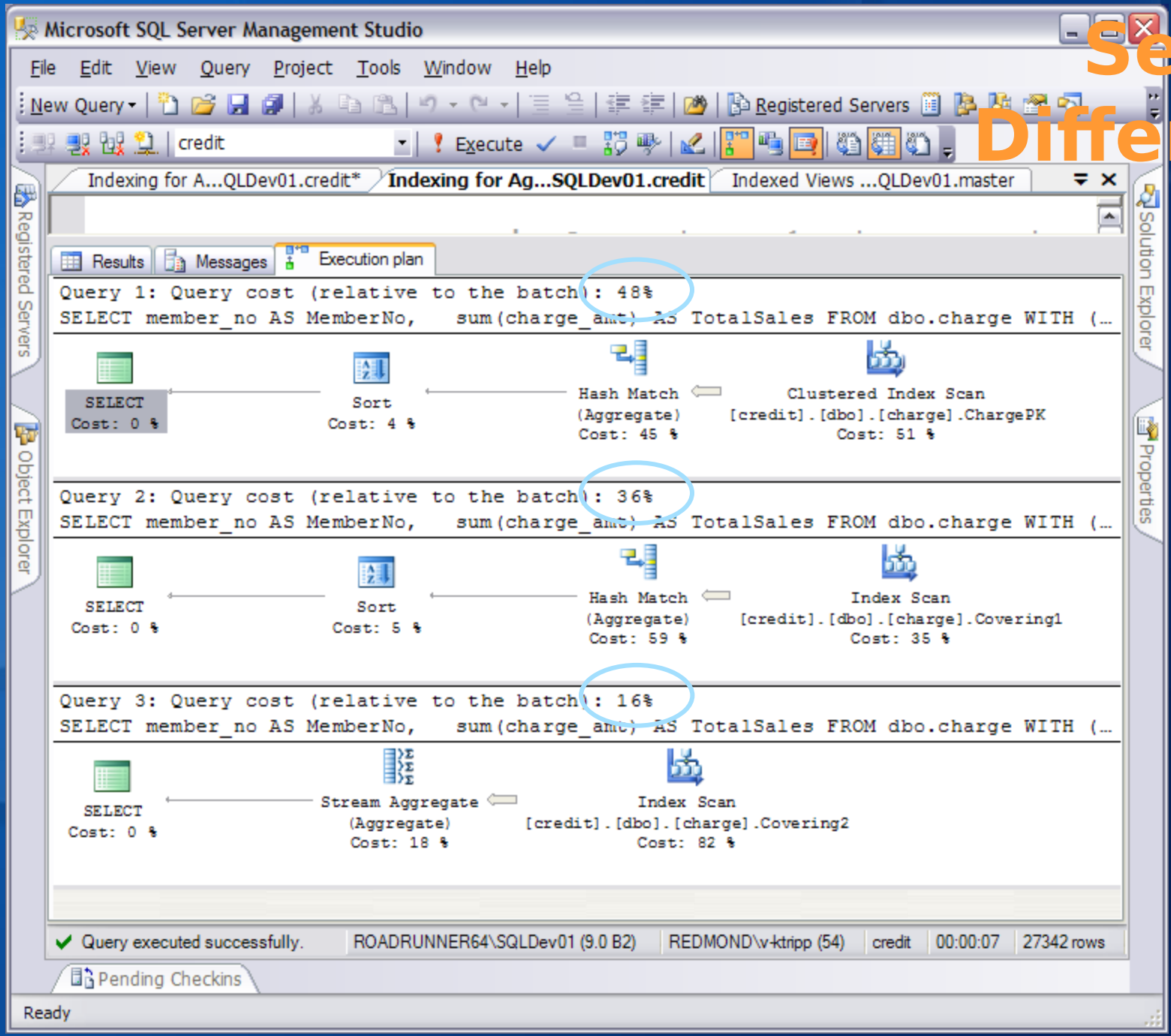
- Covering Index on Charge Table – In ORDER of GROUP BY Clause
  - Index Exists which is narrower than base table
  - Used instead of table – to cover the query
  - Covers the GROUP BY so data is grouped
- Less work to aggregate results IN ORDER
- Data Returned IN ORDER – unless ORDER BY or other joins added
- Adding an ORDER BY identical to the GROUP BY does NOT cause any additional step for sorting!

Much Better!

- COVERING Index Scan 1,600,000 narrower rows
- Stream Aggregate also yields 9,114 rows IN ORDER NO SORT REQUIRED
- Return Data



See the Difference ?



Concerns

- More temp tables
- More contention in tempdb
- Larger tempdb required
- Performance Varies on each execution
- Aggregate needs to be computed

Is there a better way?

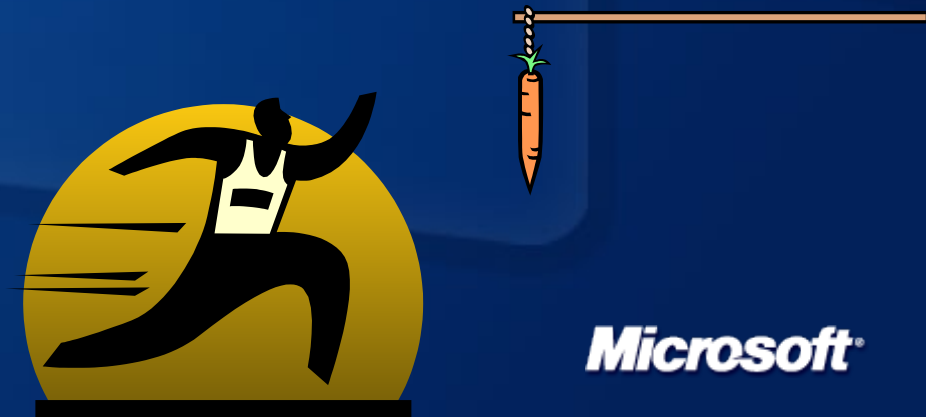
Indexed Views!

demo

The Punch Line?

What kinds of gains can you get?

Will it be worth it?



Microsoft

Aggregate Query Indexed view

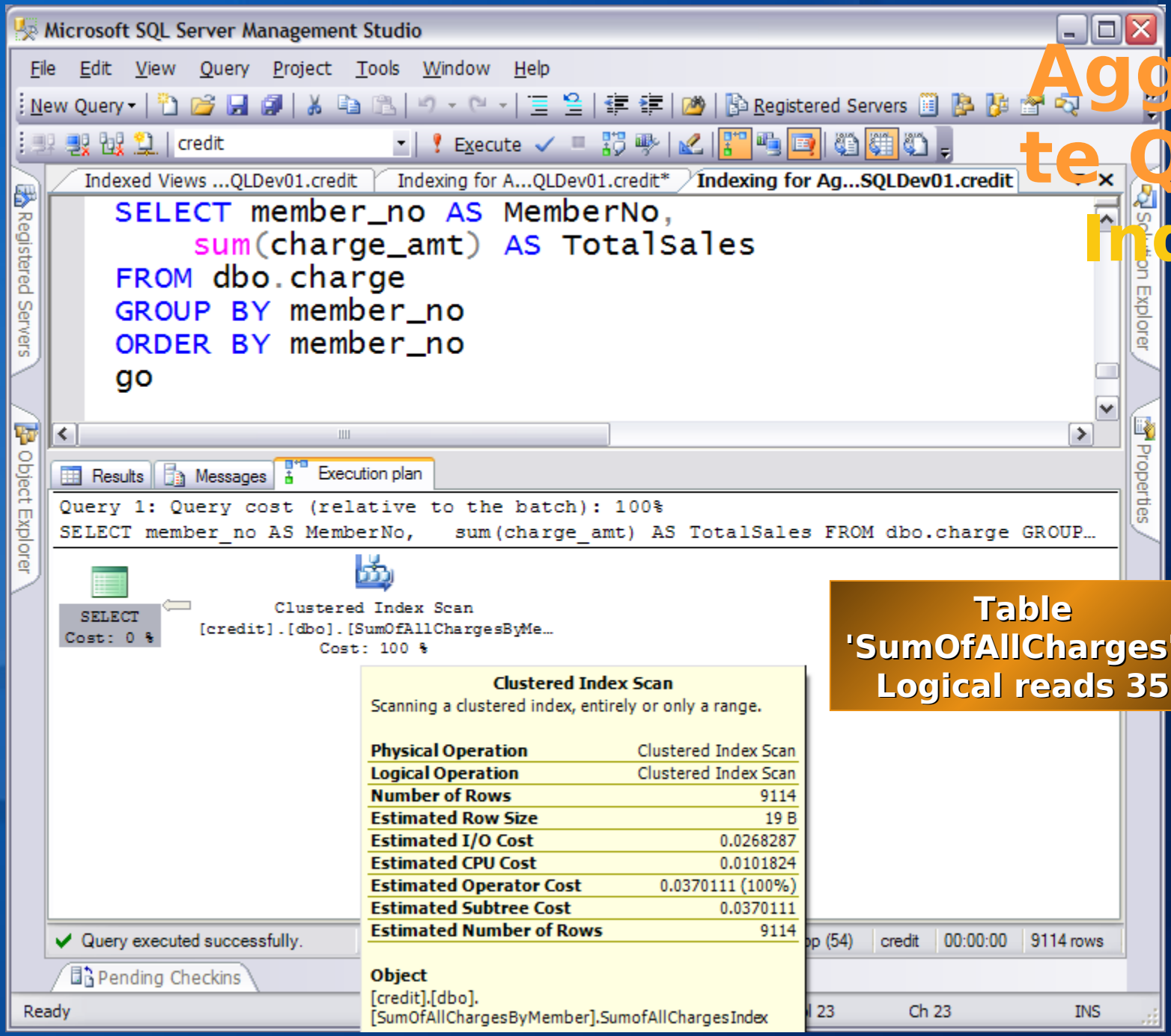
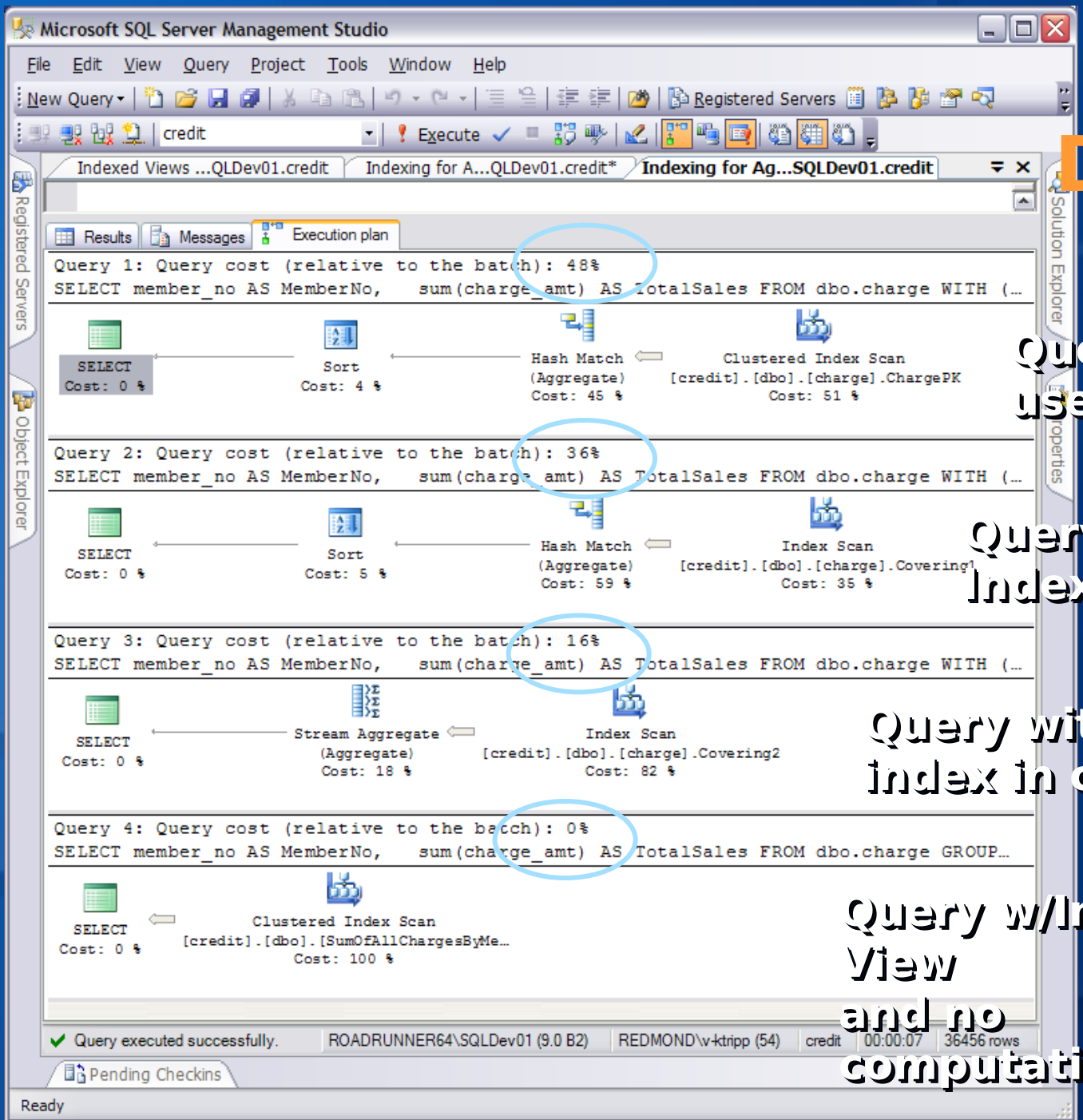


Table 'SumOfAllCharges'. Logical reads 35

See the Difference ?



Query with NO useful indexes

Query with covering index in wrong order

Query with covering index in correct order

Query w/Indexed View and no computations!

Indexed View for Aggregates
Key points

- TempDB access not necessary
- NO worktables are necessary
- Aggregated set should be small but not too small as to create a hot ROW spot of activity (which can create excessive blocking)
  - GROUP BY member_no – probably OK
  - GROUP BY state – too few rows in aggregate
  - GROUP BY country – *avoid* like the plague!
- Performance of data modification statements should be tested

INCLUDE Non-key Columns Compared to Indexed Views

- Indexed Views allow aggregates – adding interesting columns in the leaf level of an index offers “creative covering”
- With new INCLUDE clause, leaf level of index can include non-key columns
- Index key [has been since 7.0] limited to 900 bytes/16 columns – this is to keep tree structure and non-leaf levels optimal/small
- Allows more covering indexes
- Indexed View v. Include – depends on what needs to be in the leaf level

Finding the Right Balance
Index strategies: Summary

- Determine Primary Usage of Table – OLTP vs. OLAP vs. Combo? This determines Clustered Index
- Create Constraints – Primary Key and Alternate/Candidate Keys
- Manually Add Indexes to Foreign Key Constraints
- Capture a Workload(s) and Run through Index Tuning Wizard
- Add additional indexes to help improve SARGs, Joins, Aggregations

Are you done?

NO!

Keeping Performance Optimal

- Write effective queries to help optimizer use indexes
- Make sure statistics are accurate
  - Keep auto update statistics ON
  - Keep auto create statistics ON
- Make sure indexes don’t become overly fragmented
  - Rebuild Table/Index Structures
  - Defrag Table/Index Structures

More details regarding Index Fragmentation and Reindexing in SQL Server 2005 in Session DBA329 after this session!

Microsoft

Review

- Index Usage and Effective Queries
- Index Concepts
- Table Structures: Internals
- Clustered Tables: The Pros and Cons
- Non-clustered Indexes: The Pros and Cons
- Finding the Right Balance
- Adding Non-clustered Indexes for Better Performance
- Keeping performance optimal

Resources

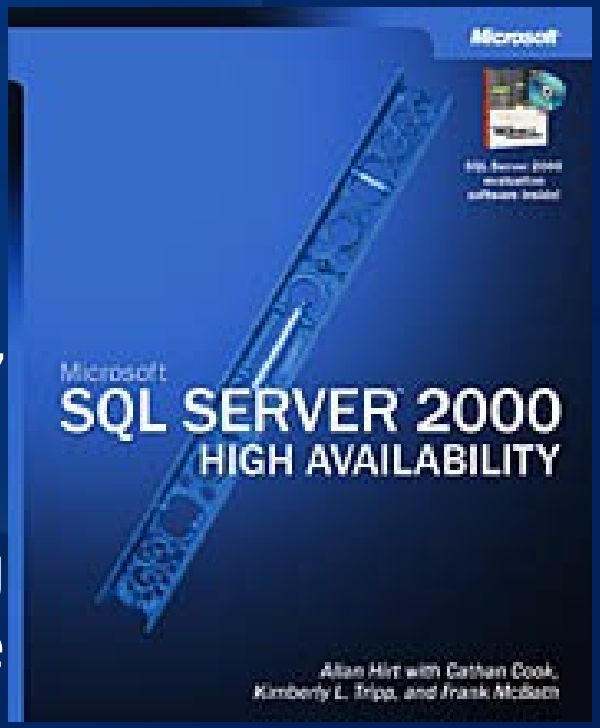
- The SQL Server 2005 Developer Center on msdn

  http://msdn.microsoft.com/SQL/2005/default.aspx
- “SQL Server 2005 Webcasts” contains links for 20+ webcasts
- “SQL Server Hands-On labs” contains links to online virtual lab environments and lab resources
- “SQL Server 2005 Articles” contains sections for different disciplines and links to 25+ articles/whitepapers
- Keep watching the Developer Center, there are new resources every week!

Microsoft

Resources

- Check out www.SQLskills.com for information about upcoming **SQL Immersion** events, useful links and event scripts.
- Read my blog: http://www.SQLskills.com/Blogs/Kimberly/
- Subscribe to SQLskills: http://www.SQLskills.com/login.aspx
- MSPress: *SQL Server 2000 High Availability*
  *A*uthors: Allan Hirt with Cathan Cook, Kimberly L. Tripp and Frank McBath
  ISBN: 0-7356-1920-4
  On the SQLskills.com homepag
  can download a sample chapte



questions?

Microsoft

Community Resources

- Microsoft Community Resources
  http://www.microsoft.com/communities/default.mspx
- Non-Microsoft Community Resources
  http://www.microsoft.com/communities/related/default.mspx
- Newsgroups
  Converse online with Microsoft Newsgroups, including Worldwide
  http://www.microsoft.com/communities/newsgroups/default.mspx
- User Groups
  Meet and learn with your peers
  http://www.microsoft.com/communities/usergroups/default.mspx
- Attend a free chat
  http://www.microsoft.com/communities/chats/default.mspx
- Attend a free web cast
  http://www.microsoft.com/usa/webcasts/default.asp
- Most Valuable Professional (MVP)

Microsoft

Microsoft Learning Resources:

Get ready for Microsoft Visual Studio 2005 and Microsoft SQL Server 2005 with free Assessments and E-Learning, and a chance to win a laptop, GPS, and more

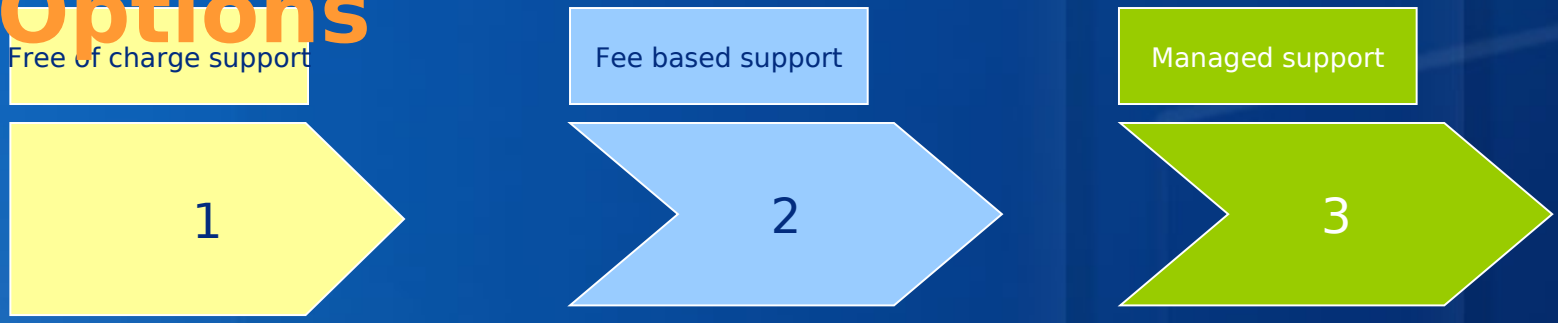
- Click here to access free Microsoft Learning Assessments http://www.microsoft.com/learning/assessment/ind/default.asp
- To access free Microsoft Learning E-Learning visit http://www.microsoft.com/learning/access and reference the promotional code **9185-TECHED-6650**.

Special offers on Microsoft Certification from Microsoft Learning

- Click here to take advantage of these special offers: http://www.microsoft.com/learning/mcp/

Microsoft

The Microsoft Guide to Support Options



Self Support

- Knowledge Base
- Microsoft Newsgroups
- Support Webcasts

Assisted Support

- Email Support
- Phone Support
- Advisory Services
  - Remotely delivered, hourly fee–based, consultative support option

Contract-based Support

- Microsoft Premier Support
  - Provides premium level support and a designated Account Manager
- Microsoft Essential Support
  - Prepackaged support options including phone and online access to a pool of skilled Technical Account Specialists
- Microsoft Partner Advantage
  - Partner Advantage provides technology partners the best response times and highest level of problem resolution support

Resources:

Microsoft Support Website: http://support.microsoft.com
Microsoft TechNet program: http://technet.microsoft.com
Microsoft MSDN program: http://msdn.micrososft.com

Microsoft


All attendees who submit a session feedback form within 120 minutes after the session ends will have the chance to win an MPx220 With Windows Mobile™ Software.
Added Bonus!! This year you will also have the chance to win one of eight Xbox 360s! We will be giving away two a day – one for the morning sessions, and one for the afternoon sessions.
Good luck!
MOTOROLA
MPx220
XBOX 360